Nivel de Transporte - Laboratorio

Teoría de las Comunicaciones

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

08.10.2014

Agenda

1. Introducción
   - Qué sabemos hasta ahora
   - Nivel de transporte: conceptos esenciales
   - El protocolo UDP
   - Flujos de datos
   - NAT con puertos

2. Discovery a nivel transporte
   - Técnicas de port scanning
   - Detección de servicios
   - Detección de sistemas operativos
   - Herramientas

Agenda

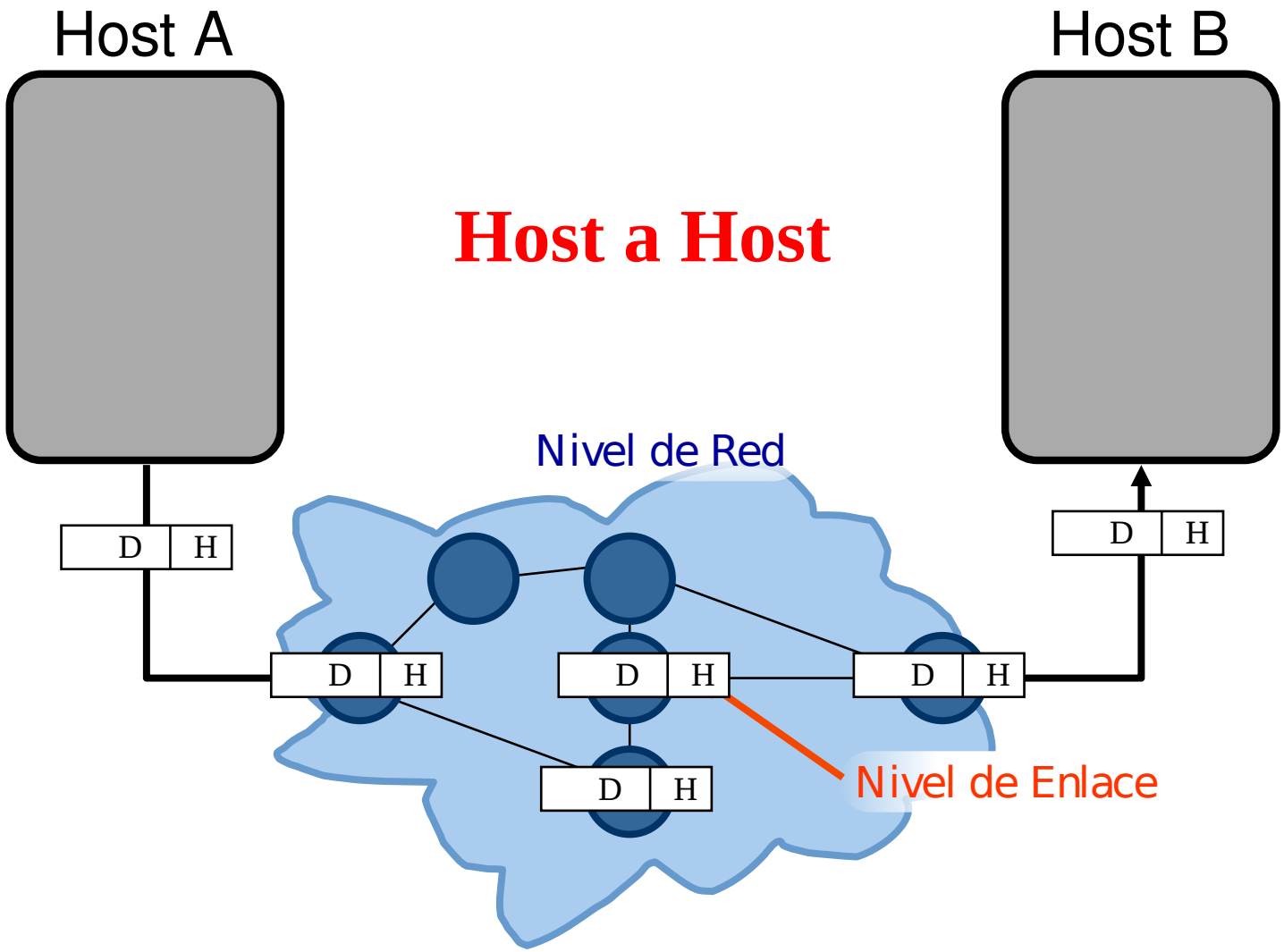
1. Introducción
   - Qué sabemos hasta ahora
   - Nivel de transporte: conceptos esenciales
   - El protocolo UDP
   - Flujos de datos
   - NAT con puertos

2. Discovery a nivel transporte
   - Técnicas de port scanning
   - Detección de servicios
   - Detección de sistemas operativos
   - Herramientas

Los niveles conocidos del modelo OSI

- **Nivel físico**: lidia con el problema de cómo transmitir bits en un medio dado.
- **Nivel de enlace**: conecta dispositivos en una misma red local y transfiere tramas (frames) entre ellos.
- **Nivel de red**: define cómo interconectar redes y dirigir el tráfico de paquetes entre sus destinos.

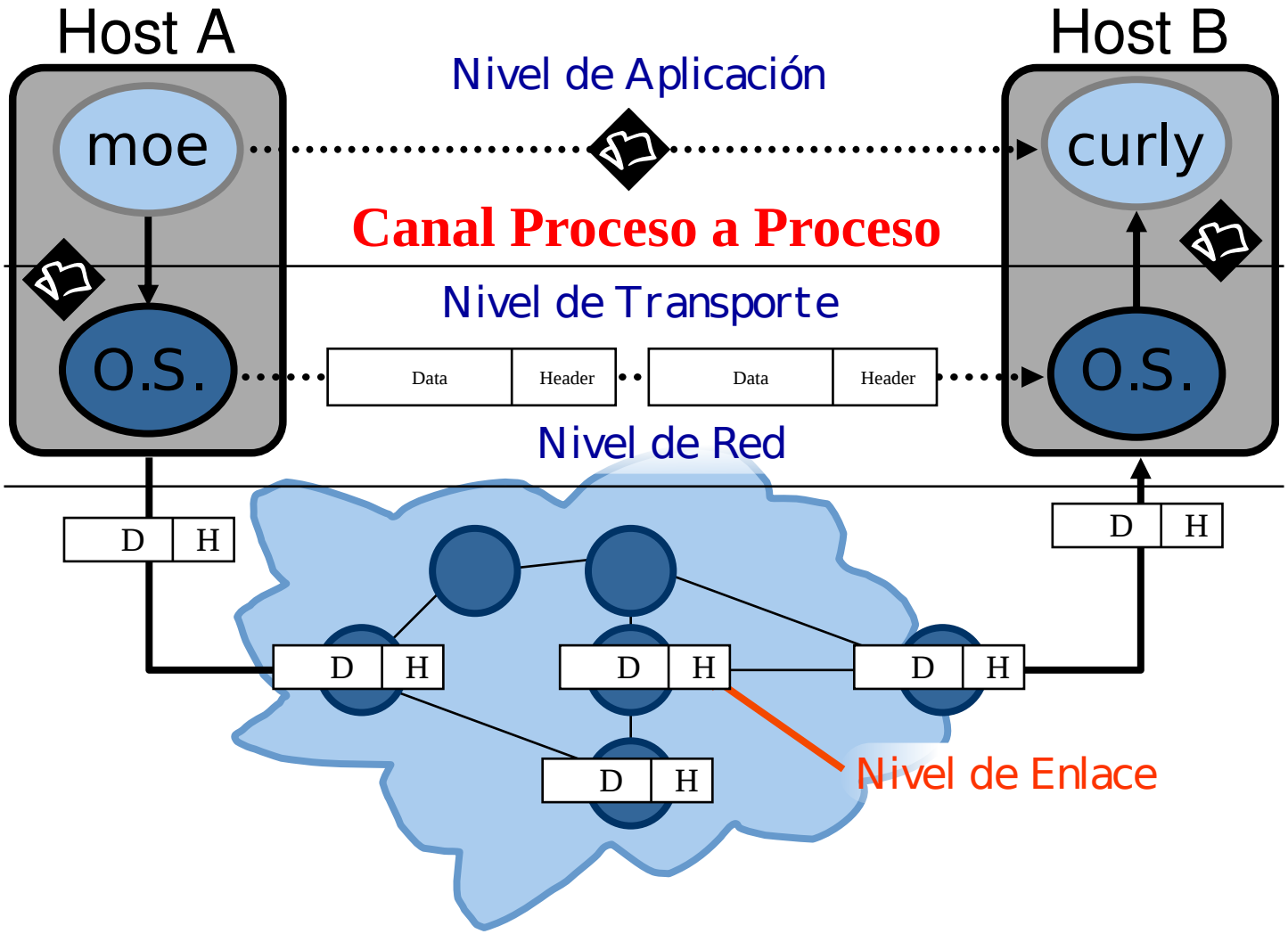
El nivel de red y sus amigos ilustrados



¿Para qué está el nivel de transporte?

- En resumen, hasta ahora tenemos un mecanismo para comunicar hosts con hosts.
- Ahora, el objetivo está en implementar un canal de comunicación entre **procesos**.
- Este canal será en definitiva utilizado por las aplicaciones.
- Problema: capas inferiores suelen tener limitaciones y las aplicaciones pueden requerir servicios confiables.

El nivel de transporte ilustrado



Características esenciales

- El nivel de red ofrece un servicio de *mejor esfuerzo*:
  - Puede descartar mensajes,
  - Reordenar mensajes,
  - Entregar copias duplicadas de un mensaje,
  - Limitar los tamaños de mensaje a un valor fijo, o
  - Entregar mensajes con una demora arbitraria.
- Sin embargo, el nivel de transporte, para satisfacer a las aplicaciones, debe ofrecer:
  - Entrega de mensajes garantizada y respetando el orden,
  - No entregar mensajes duplicados,
  - Soportar tamaños de mensaje arbitrarios,
  - Control de flujo por parte del emistor hacia el receptor,
  - Soportar múltiples procesos en cada host, entre otras.
- $\Rightarrow$ **Objetivo**: desarrollar algoritmos para alcanzar estos requerimientos.

Esto me suena de otro lado...

- De lo anterior puede pensarse que el nivel de transporte guarda similitudes con el nivel de enlace:
  - Ambos deben lidiar con control de errores, secuenciamiento, control de flujo, etc.
- **Pero** el nivel de enlace se apoya en un canal físico y el nivel de transporte tiene debajo una red:
  - Se requiere **direccionamiento** explícito,
  - Establecer conexiones es indefectiblemente más complicado,
  - La red posee *capacidad de almacenamiento*: un paquete puede quedar almacenado en la red y llegar a destino más tarde,
  - Etc.

Cómo identificamos los procesos

- Al iniciar la comunicación, los procesos deben especificar con qué proceso destino desean hablar.
- No sirve PIDs: scope local, y son variables.
- La arquitectura TCP/IP utiliza **puertos**: valor numérico (de 16 bits) que identifica procesos.
- Se utiliza junto con la dirección IP para dar una caracterización unívoca del destino.
- Usualmente implementados mediante **sockets**.

Puertos: cómo funcionan

- Cuando un proceso se asocia a un puerto, está listo para recibir mensajes.
- Problema: los procesos externos deben tomar conocimiento de este puerto elegido.
- Soluciones:
  - Utilizar puertos **conocidos**: sucede en la práctica. Ejemplo: web server en puerto 80, mail server en puerto 25, etc.
  - Utilizar un **port mapper**: proceso que sabe cuál puerto utiliza cada servicio. Corre en un puerto fijo. Ejemplo: RPC.
- Los servidores saben cómo responder a los clientes: el puerto respectivo viene en los mensajes que ellos envían.

Puertos y servicios en Linux

- En Linux, el archivo `/etc/services` lista los puertos conocidos de los distintos servicios:

```
ftp-data        20/tcp
ftp             21/tcp
fsp             21/udp          fspd
ssh             22/tcp
ssh             22/udp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
```

- Se usa para obtener, desde el código, el puerto de un servicio a través de su nombre.

Ejercicio: puertos y servicios desde Python

Las funciones `getservbyname` y `getservbyport` permiten obtener, respectivamente, el puerto "default" asociado a un servicio dado y el nombre del servicio asociado a un puerto dado.

1. En Python, importar el módulo `socket` y obtener el servicio asociado al puerto 80.
2. Encontrar por lo menos un puerto `p` tal que `socket.getservbyport(p)` esté definido y `socket.getservbyname(socket.getservbyport(p)) != p`. ¿A qué se debe esto?
3. Agregar en `/etc/services` un servicio `foo` asociado al puerto `61000/tcp` y comprobar que `getservbyname` y `getservbyport` lo reconocen. ¿Qué sucede si cambiamos el puerto por el puerto de un servicio ya existente en el archivo?

Clasificación de protocolos de transporte

- Podemos clasificar los protocolos de nivel de transporte a través de las siguientes características:
  - Orientación a conexión
  - Confiabilidad
- En TCP/IP se distinguen fundamentalmente estos protocolos:
  - **TCP**: con conexión y confiable (visto en la clase de ayer).
  - **UDP**: sin conexión y no confiable (más detalles en lo que viene).
- Hay otros. Ejemplo: RUDP (UDP extendido para hacerlo confiable y al mismo tiempo evitar el overhead de TCP).

- UDP -

El protocolo UDP

- UDP (*User Datagram Protocol*), como hemos dicho, es no confiable ni orientado a conexión.
- Especificado en el RFC 768.
- Extiende el servicio de la capa de red subyacente transformándolo en un canal de comunicación entre procesos.
- $\Rightarrow$ Agrega **demultiplexación** a IP.

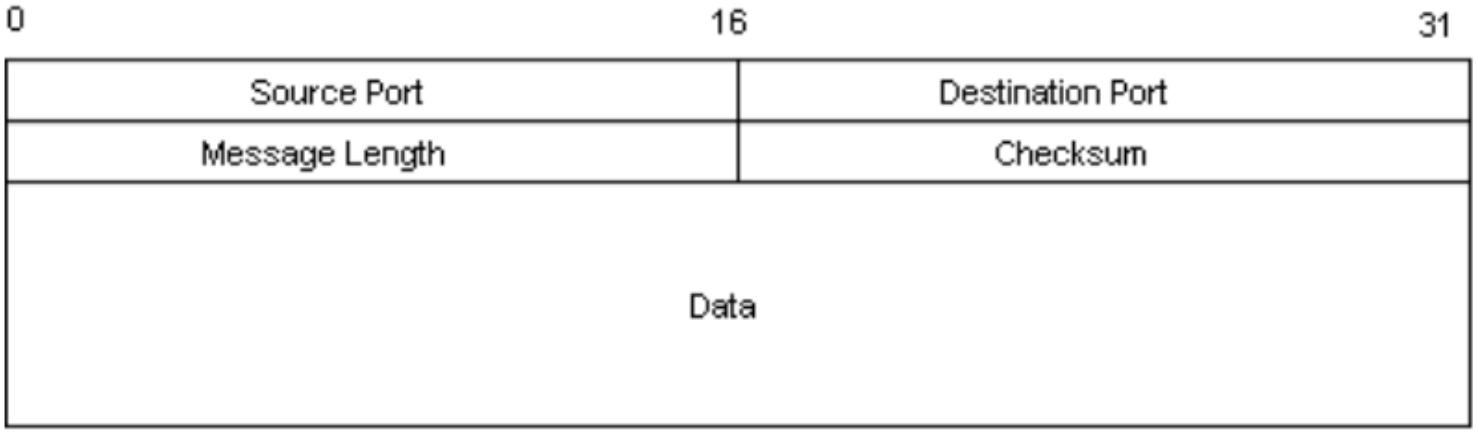
Características de UDP

- Establece una comunicación poco costosa entre procesos.
- Evita la sobrecarga y las demoras propias de la entrega ordenada y confiable de mensajes.
- Soporta múltiples procesos de nivel de aplicación en cada máquina.
- Tráfico *egoísta*: puede congestionar las colas de entrada y salida de los routers.
- Sin control de flujo: el destinatario puede verse saturado y perder información en consecuencia.
- Esencialmente, **IP** + **puerto**.

¿Por qué UDP?

- Control preciso sobre qué información se envía y cuándo se envía.
- Sin demoras para establecimiento de conexiones.
  - UDP no contempla ningún procedimiento preliminar formal.
- Sin *estado de conexión*: no se reservan parámetros, números de secuencia, etc.
  - Esto facilita la manipulación de muchos clientes activos en simultáneo.
- Poco overhead por encabezado: sólo 8 bytes.

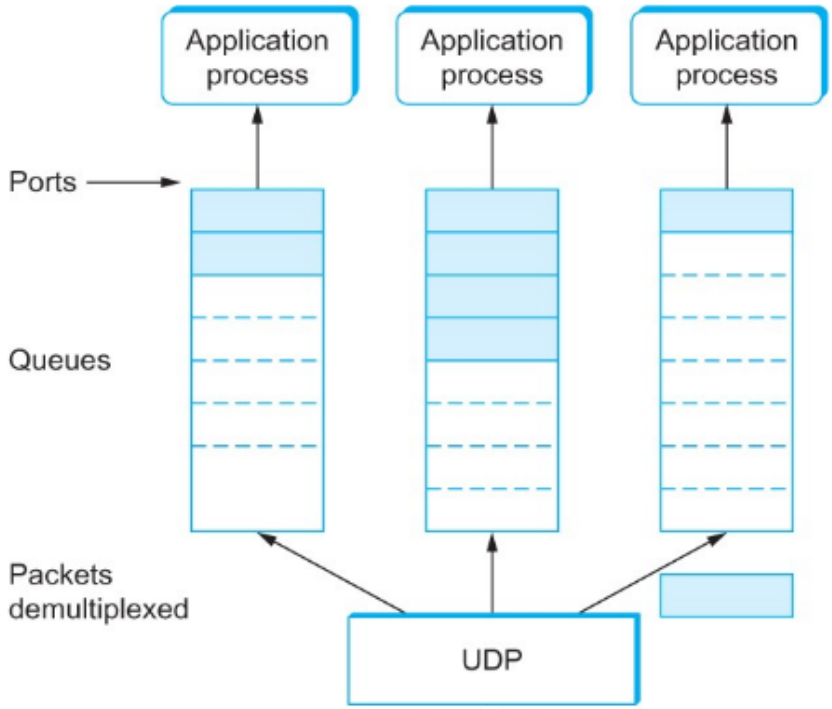
Formato del datagrama



- `Message Length`: longitud (en bytes) del datagrama completo. Tamaño mínimo: 8 bytes (¿por qué?).
- `Checksum`
  - Se computa sobre el header, los datos y el *pseudoheader*: campos `Protocol`, `Source Address` y `Destination Address` del header IP.
  - Opcional en IP; obligatorio en IPv6.

Puertos como colas de mensajes (cortesía Peterson)

- Cuando un paquete arriba a destino, éste se dirige al puerto correspondiente y aguardará allí encolado.
- De no haber espacio suficiente, se descartará sin más.
- La aplicación removerá el primer paquete encolado al momento de recibir mensajes.
- Si la cola está vacía, se bloqueará.

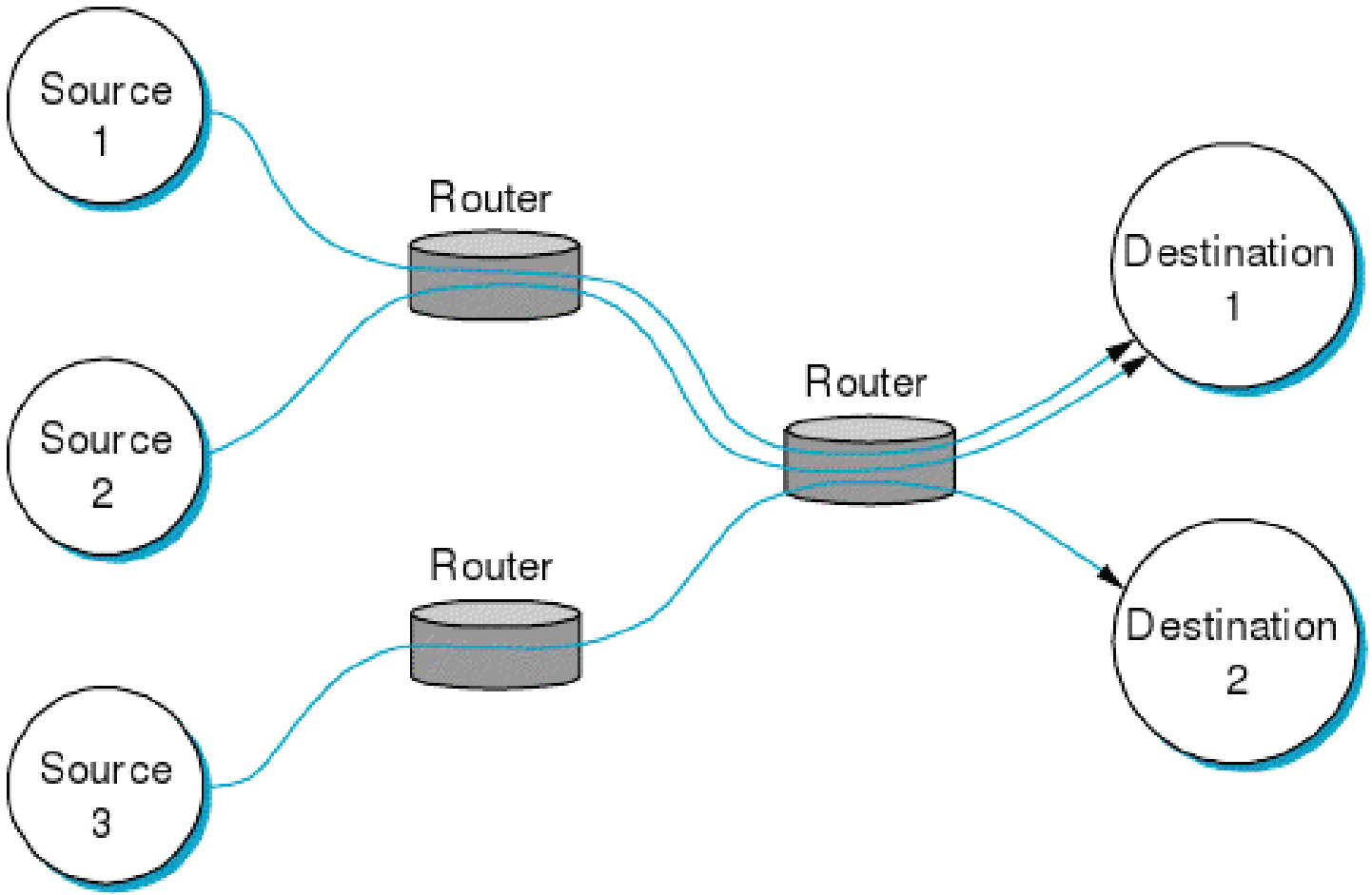


- Flujos de datos -

Flujos de datos

- Un *flujo de datos* es una secuencia de paquetes entre un mismo origen y destino que sigue una misma ruta a través de la red.
- Puede tener distintas granularidades:
  - Host a host: usa sólo direcciones de origen y destino.
  - Proceso a proceso: involucra también los puertos.
- Similar en este caso al concepto de canal.
- Pero visible por cada router!
- Abstracción importante en control de congestión y reserva de recursos en routers.

Flujos ilustrados (cortesía Peterson)



Taxonomía de flujos

- **Sin conexión**: si bien los paquetes son independientes entre sí, puede ocurrir que varios pasen por el mismo conjunto de routers. Ejemplo: Internet.
- **Con conexión**: en la fase de inicio de conexión se reservan recursos en los routers intermedios. Ejemplo: circuitos virtuales.

Soft state

- Es un punto intermedio entre ambos extremos.
- Si bien no existe una etapa explícita de inicialización, un router puede mantener cierta información sobre un flujo.
- Esto puede colaborar en la reserva de recursos para los paquetes pertenecientes a dicho flujo.
- Observar que el ruteo **no depende** de la presencia de esta información!

Flujos en el modelo TCP/IP

- Pueden verse como una tupla
  $\langle$IP origen, Puerto origen, IP destino, Puerto destino, Protocolo$\rangle$
- en donde Protocolo indica el protocolo de nivel de transporte utilizado (i.e., TCP o UDP).
- Los routers, a partir de estas tuplas, pueden generar clases de equivalencia de ruteo para paquetes en distintos flujos. Por ejemplo,
  $\langle 200.100.25.0/24, *, 200.100.26.101, 80, \text{TCP}\rangle \overset{\text{Int. } 1}{\rightarrow} 200.100.26.5$

Flujos en el mundo real

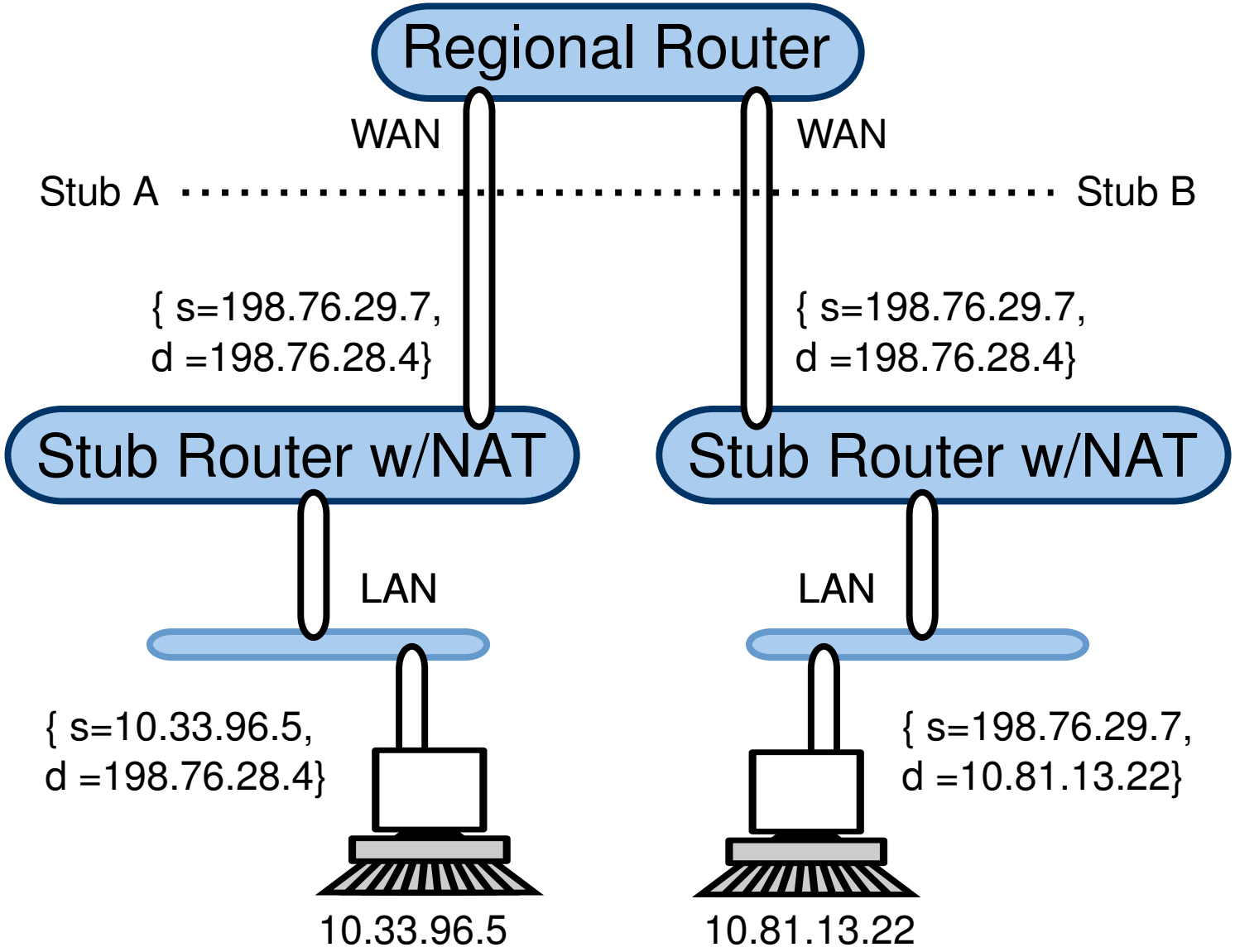
- Existen diversas tecnologías para monitorear y analizar flujos en las redes modernas.
- Una de ellas es **NetFlow**, de Cisco.
  - Cada router con **NetFlow** habilitado colecta tráfico entrante y saliente e identifica los distintos flujos.
  - Una vez finalizado un flujo, exporta en registros la información obtenida.
  - Los registros son enviados un servidor, el **NetFlow collector**.
  - Es éste quien realiza el análisis de los flujos.
  - Luego, un administrador puede conectarse al collector y realizarle consultas.
- Otra es **sFlow** que, a diferencia de la anterior, trabaja a nivel 2.

- NAT con puertos -

¿Qué era eso de NAT?

- NAT (*Network Address Translation*) es un método para mapear direcciones IP de un dominio (usualmente privado) a otro (usualmente público).
- Su forma más básica está especificada en el RFC 1631.
- Función implementada en los gateways: transparente para los hosts.
- Se ideó para atacar el problema de la escasez de direcciones IP.
- Pools de direcciones privadas pueden reutilizarse arbitrariamente.

NAT ilustrado (cortesía RFC)



Detalles...

- NAT no sólo modifica la dirección IP de los paquetes.
- **Debe** modificar el checksum.
- ...y también el checksum de nivel de transporte por el pseudoheader!
- En general, debe tocar cualquier parte donde aparezca la IP.
- Ejemplo: ICMP (payload).
- Potenciales problemas:
  - Aplicaciones podrían romperse.
  - Seguridad.

NAT con puertos (o NAPT)

- Una extensión: NAPT (*Network Address and Port Translation*).
- Mapeo de muchos-a-uno a diferencia del mapeo biyectivo anterior.
- La dirección pública es la misma para todos los hosts de la red privada.
- Los puertos (públicos) son los que identifican el tráfico y definen el mapeo.
- Terminología en el RFC 2663.

Cómo trabaja NAPT

- Cuando un host quiere contactar un servicio externo, el paquete saliente de su red privada contendrá también ahora un puerto origen **distinto**.
- El gateway realiza esto y guarda la entrada en su tabla de mapeos.
- Al revés no: un host externo no puede iniciar la comunicación.
- Solución: **port forwarding**.
  - Reglas definidas en el gateway para asociar un puerto externo a un servicio interno.
  - Ejemplo: (190.172.12.16, 8080) → (192.168.0.100, 80)

Demo/ejercicio: observando tráfico NATeado

- Para bajar esto a la tierra, veamos en vivo cómo trabaja NAT.
- De paso, vamos a usar `ping` (i.e., ICMP) para analizar cómo hace NAT para mapear protocolos sin puertos.
- Lo haremos siguiendo, esencialmente, estos pasos:
  1. Prender una máquina virtual configurada en red privada con nuestro host y agregarle como *default gateway* la IP (privada) de nuestro host.
  2. Configurar el SO del host para que haga forwarding (i.e., que se comporte como un router) y para que haga NAT.
  3. Hacer `ping` a un IP arbitraria desde la VM y capturar el tráfico con Wireshark/`conntrack`.

Primer paso: configurando la VM

- Típicamente, el host tendrá definida una interfaz de red virtual creada por el software de virtualización.
- Usar `ifconfig` para determinar cuál es (ya que la vamos a necesitar).
- En el caso del ejemplo, esta interfaz se llama `vmnet1` y su IP es `192.168.203.1`.
- ⇒ Debemos decirle a la VM que su default gateway es `192.168.203.1`:

```
$ sudo route add default gw 192.168.203.1
```

Segundo paso: configurando el host

- Ahora, digámosle al kernel de Linux (de nuestro host) que habilite el forwarding de datagramas IPv4:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

- Y usemos `iptables` para activar NAT:

```
$ sudo iptables -table nat -delete-chain
$ sudo iptables -t nat -A POSTROUTING -o wlan0 -j
MASQUERADE
$ sudo iptables -A FORWARD -i vmnet1 -j ACCEPT
```

- En este caso, la interfaz de salida (i.e., la interfaz real del host por la que obtiene conectividad a Internet) es `wlan0`.
- Al igual que `vmnet1`, este valor también podría cambiar si se quisiera repetir el experimento en otra plataforma.

Tercer paso: haciendo `ping` a Google desde la VM

- En una consola de la VM, hacer `ping` a una IP arbitraria (e.g., `173.194.42.241`, de Google):

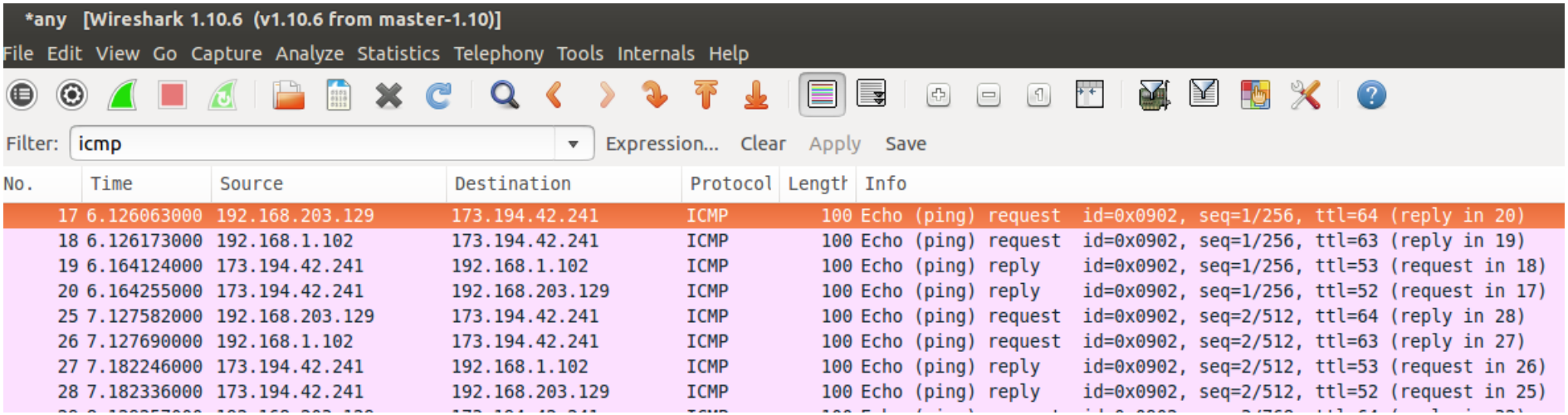
```
$ ping 173.194.42.241
```

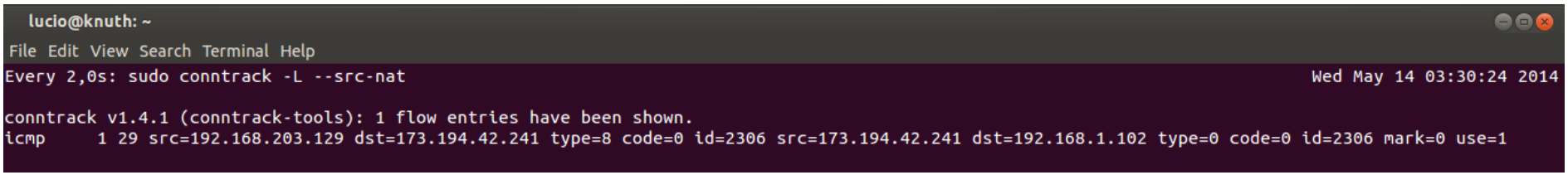
- En el host, correr Wireshark y capturar el tráfico ICMP. Observar qué ocurre.
- Adicionalmente, instalar `conntrack` y ver qué conexiones están siendo NATeadas por el kernel:

```
$ sudo conntrack -L -src-nat
```

- Resultados en las próximas slides (por si la demo en vivo falla ☺).

Resultados: capturas de pantalla de Wireshark y conntrack





Agenda

1. Introducción
   - Qué sabemos hasta ahora
   - Nivel de transporte: conceptos esenciales
   - El protocolo UDP
   - Flujos de datos
   - NAT con puertos

2. Discovery a nivel transporte
   - Técnicas de port scanning
   - Detección de servicios
   - Detección de sistemas operativos
   - Herramientas

Recopilación de info - *information gathering*

- Desde la óptica de usuario de red arbitrario, hasta ahora sabemos cómo reconocer hosts activos y al alcance en una red desconocida.
- El siguiente paso:
  - Determinar puertos disponibles,
  - Determinar servicios disponibles (y sus respectivas implementaciones), e
  - Identificar el sistema operativo subyacente.
- Permite, por ejemplo, armar un inventario de la red.
- Además, tiene implicancias importantes en seguridad informática.

A por los puertos!

- La identificación de los puertos disponibles se conoce como *port scanning*.
- Procedimiento que barre una secuencia de puertos en un host dado enviando paquetes y analizando las posibles respuestas.
- Se puede materializar de formas variadas, según la heurística que se desee utilizar.
- Hoy veremos:
  - SYN scanning,
  - Connect scanning,
  - Xmas, null y FIN scanning, y
  - UDP scanning.
- Se asume (en los primeros) compatibilidad con la especificación de TCP (RFC 793).

Clasificación de puertos

- Usulamente, un scan clasificará a un puerto de la siguiente manera:
- **open** (abierto) si pudo inferir de su heurística que hay un servicio escuchando en dicho puerto.
- **closed** (cerrado) si determinó que no hay un proceso escuchando allí.
- **filtered** (filtrado) si hay chances de que alguna entidad intermedia (e.g., firewall) descarte el tráfico correspondiente.
- Puede ocurrir que la heurística del scanner no pueda discernir completamente entre estas categorías.

SYN scan (o half-open scan)

- El scan más popular!
- Envía un paquete TCP con flag SYN y espera la respuesta:
  - SYN/ACK: indica que el puerto está abierto.
  - RST: indica que el puerto está cerrado.
  - Si no hay respuesta, se asume filtrado. Ídem si la respuesta es ICMP de tipo *destination unreachable*.
- No se completa la conexión.

SYN scan: ventajas y desventajas

- Suele ser rápido.
- Poco invasivo y discreto, al no completar las conexiones.
- Por otro lado, requiere enviar y recibir paquetes ad-hoc, lo cual requiere permisos elevados en el sistema.

Connect scan (o TCP scan)

- Usa la API del sistema operativo para establecer una conexión: primitiva `connect` de sockets.
- Así se inicia una conexión usualmente en clientes web, clientes de mail, clientes FTP, etc.
- Observar que se instancia por completo el algoritmo de three-way handshake!

Connect scan: ventajas y desventajas

- No requiere permisos elevados para correr.
- Pero tiene varias contras...
  - Más lento y con más volumen de tráfico, al completar conexiones.
  - Menos discreto: los hosts posiblemente registren en logs estas conexiones.

Scans bizarros (Xmas scan, null scan y FIN scan)

- Se apoyan en sutilezas del RFC.
- Si el estado de un puerto es CLOSED, la especificación dice (pág. 65):

```
 An incoming segment containing a RST is discarded.
An incoming segment not containing a RST causes a RST to
be sent in response.
```

- Si es LISTEN, afirma que todo paquete sin flags ACK, SYN ni RST (situación muy poco frecuente) **debe ser descartado** (pág. 66).

Cómo funcionan

- Si el host destino implementa coherentemente el RFC, enviar un paquete TCP que **no** incluya los flags SYN, ACK o RST resultará en:
  - Puerto cerrado ⇒ respuesta con flag RST.
  - Puerto abierto ⇒ respuesta inexistente.
- Los otros flags (PSH, URG y FIN) pueden tomar valores arbitrarios, lo cual determina las variantes de los scans:
  - Xmas scan: todos prendidos (como un arbolito de navidad!).
  - Null scan: ninguno prendido.
  - FIN scan: sólo FIN prendido.
- Observar que una falta de respuesta también puede indicar que el puerto está filtrado!

Xmas, null y FIN scans: ventajas y desventajas

- Pueden escabullirse mejor que un SYN scan a través de firewalls.
- Sin embargo, pueden resultar más sospechosos.
- Además, no es confiable con SOs que no implementen al pie de la letra estos detalles. Ejemplo: Windows.
- Y, como ya vimos, no hay una distinción clara entre puertos abiertos y filtrados.

UDP scan

- Es más complicado hacer scans por UDP: protocolo muy simple y sin conexiones!
- Pero es necesario dado que hay servicios que lo usan (como DNS o DHCP).
- Funciona enviando un paquete UDP al puerto destino:
  - Si la respuesta es ICMP de tipo *port unreachable* $\Rightarrow$ puerto cerrado.
  - Si no hay respuesta $\Rightarrow$ puede estar abierto o filtrado.
- En algunos casos pueden regresar paquetes UDP como respuesta, lo cual indica que el puerto sí está abierto.

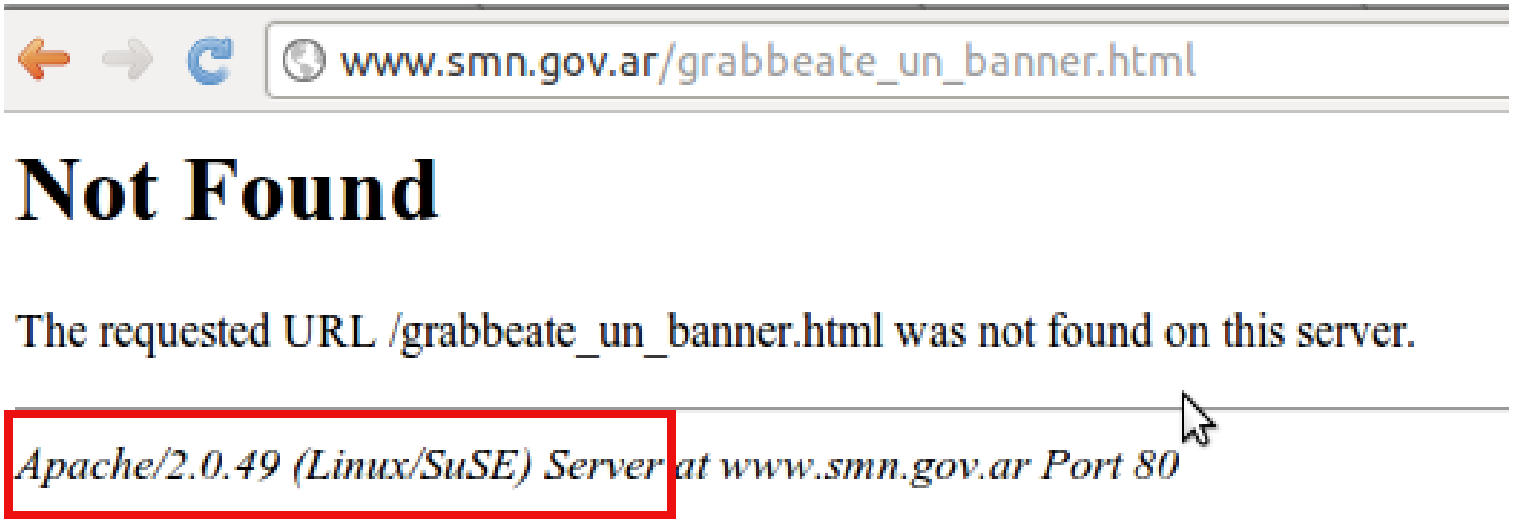
UDP scan: dificultades

- No es fácil hacerlo rápido: requiere retransmisiones por si los paquetes se extraviaron.
- Además, los hosts a veces restringen la cantidad de mensajes ICMP de tipo *port unreachable*.

Banner grabbing

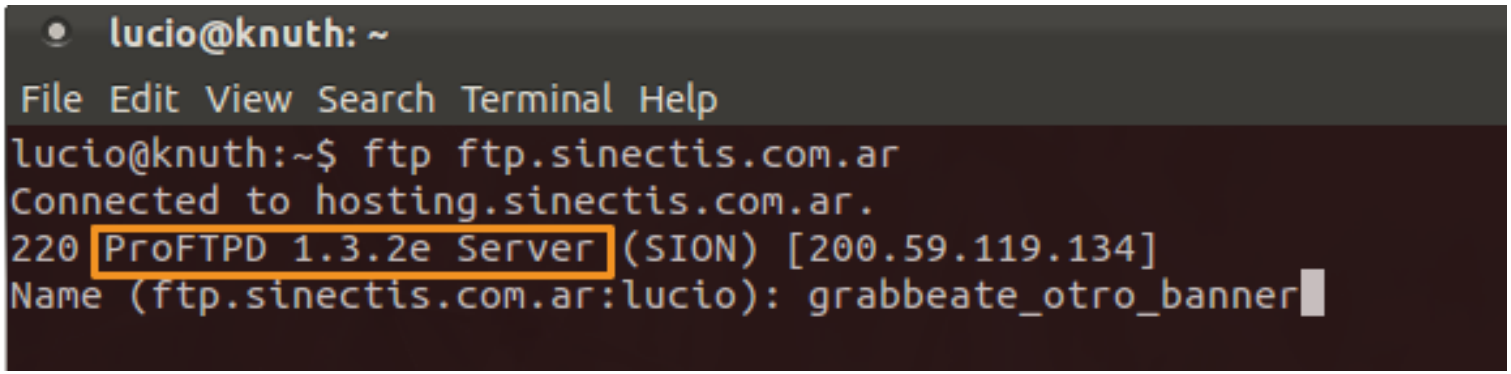
- Para identificar fehacientemente las aplicaciones detrás de los puertos, una técnica posible es *banner grabbing*.
- Consiste en conectarse al servicio y observar en la información intercambiada si hay trazas del nombre y/o versión del programa en cuestión.
- Para ello, pueden utilizarse distintas herramientas: `telnet` , `netcat` , web browsers, etc.

Banner grabbing: un ejemplo



- Observar que en este caso obtenemos también gratis el sistema operativo!

Banner grabbing: otro ejemplo



OS fingerprinting

- Ésta es la técnica para identificar la “huella característica” de un sistema operativo.
- Una forma posible de hacerlo es analizando cómo el host reacciona ante distintos paquetes.
- Por lo general se usan paquetes TCP o ICMP para los cuales las respuestas particulares de distintos sistemas operativos pueden variar, aprovechando ambigüedades en las especificaciones.
- Como vimos recién, un banner también puede darnos información acerca del sistema operativo subyacente.

Variantes

- Esencialmente existen dos variantes: detección **activa** y detección **pasiva**.
- En la primera, se envía un conjunto de paquetes al host destino y luego se procesan las respuestas obtenidas.
- La otra alternativa, más discreta, escucha la red y captura el tráfico, realizando el procesamiento posterior del mismo de manera similar a su contraparte activa.
- `nmap` es el ejemplo paradigmático de detección de SO activa.
- La herramienta `p0f` implementa detección pasiva.
- Y otra popular, `SinFP`, provee sendas estrategias.

Cómo funciona (a alto nivel)

- Usualmente, se dispone de una base de datos almacenando los posibles resultados de distintos tests para una serie de SOs.
- Luego de enviar paquetes con ciertas características, los tests consisten en estudiar los campos de las respuestas y utilizar la base para calcular el SO más probable.
- Algunos posibles tests:
  - Análisis de números de secuencia de TCP.
  - Análisis del campo ID en el header IP.
  - Análisis de opciones TCP (orden y disponibilidad variables).
- Más info acá: `http://nmap.org/book/osdetect-methods.html`

- Herramientas -

telnet

- En realidad es un protocolo de nivel de aplicación (sobre TCP)...
- Está especificado en el RFC 854.
- Provee acceso a una interfaz de línea de comandos en la máquina destino.
- Muy usado, no obstante, para testear conectividad.
- Hoy por hoy su uso para acceso remoto quedó obsoleto por ser poco seguro (a diferencia de SSH).

telnet : un ejemplo

- Veamos cómo usar telnet para conectarnos a un servidor de mails:

```
lucio@knuth:~$ telnet proxymail1.sion.com 25
Trying 200.81.186.15...
Connected to proxymail1.sion.com.
Escape character is '^]'.
220 mx2.sion.com ESMTP (SION)
QUIT
221 2.0.0 Bye
Connection closed by foreign host.
```

netstat

- Sirve para ver y analizar las conexiones TCP (...y UDP).
- Puede indicar los procesos ligados a cada socket.
- Muestra el estado de las conexiones TCP (según el famoso diagrama de estados que ya conocemos).

netstat : fragmento de una corrida

```
lucio@knuth:~$ netstat -an | more
(...)
Proto   Local Address          Foreign Address          State
tcp     127.0.0.1:631          0.0.0.0:*                LISTEN
tcp     127.0.0.1:5432         0.0.0.0:*                LISTEN
tcp     0.0.0.0:902            0.0.0.0:*                LISTEN
tcp     0.0.0.0:80             0.0.0.0:*                LISTEN
tcp     192.168.1.101:46670    65.55.71.176:1863        ESTABLISHED
tcp     192.168.1.101:37853    74.125.130.125:5222      ESTABLISHED
tcp     192.168.1.101:57385    192.168.1.100:22         ESTABLISHED
(...)
```

netcat

- La navaja suiza para TCP/IP.
- Múltiples y variadas funcionalidades:
  - Hablar con servidores (como hicimos con `telnet`)
  - Escaneo de puertos
  - Transferencia de archivos
  - Escuchar en un puerto dado

netcat : ejemplo de port scanning

```
lucio@knuth:~$ nc -vz 192.168.1.100 77-83
nc: connect to 192.168.1.100 port 77 (tcp) failed
nc: connect to 192.168.1.100 port 78 (tcp) failed
nc: connect to 192.168.1.100 port 79 (tcp) failed
Connection to 192.168.1.100 80 port [tcp/www] succeeded!
nc: connect to 192.168.1.100 port 81 (tcp) failed
nc: connect to 192.168.1.100 port 82 (tcp) failed
nc: connect to 192.168.1.100 port 83 (tcp) failed
```

netcat : ejemplo de transferencia de archivos

```
lucio@knuth:~$ nc -l localhost 12345 > file &
[1] 5557
lucio@knuth:~$ echo "eh amigo" > eh_amigo
lucio@knuth:~$ nc localhost 12345 < eh_amigo
[1]+  Done                    nc -l localhost 12345 > file
lucio@knuth:~$ cat file
eh amigo
```

ss

- *socket statistics*: herramienta para investigar el estado interno de los sockets.
- Similar a `netstat` , aunque permite obtener información de las variables internas de TCP:
  - Estimación actual del RTT y su varianza (RTTVAR)
  - Estimación actual del RTO
  - Información sobre las variables de control de congestión
  - Cantidad de retransmisiones y de segmentos no ACKeados
  - Etc.
- Además de la man page, más info (aunque no mucha) en: `http://www.cyberciti.biz/files/ss.html`

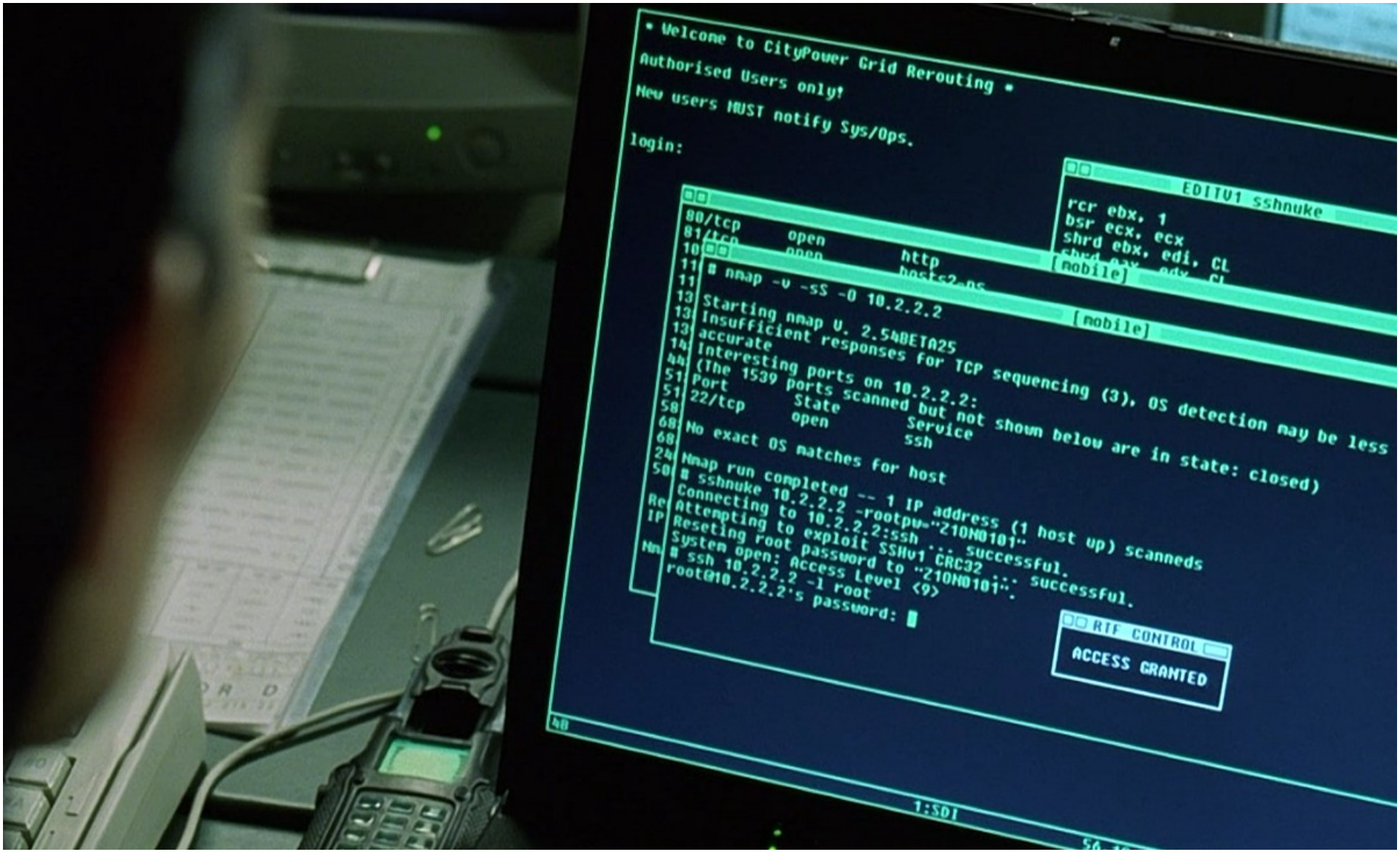
ss : ejemplo de la demo

```
Recv-Q Send-Q     Local Address:Port       Peer Address:Port
0      0              127.0.0.1:33088         127.0.0.1:5555
         cubic wscale:7,7 rto:220 rtt:21.5/10 mss:65483
         cwnd:27 ssthresh:26 send 657.9Mbps rcv_space:43690
```

nmap

- Es una herramienta de discovery muy famosa y muy completa:
  - Discovery de hosts,
  - Escaneo de puertos (implementa todas las técnicas mencionadas hace un rato y algunas otras más esotéricas),
  - Detección de servicios,
  - Detección de sistemas operativos.
- Además tiene soporte para scripts custom.
- Apareció por 1997 y tiene soporte para múltiples plataformas.
- El nombre viene de *network mapper*.

...y lo usa Trinity en Matrix!



nmap : fragmento de una corrida

```
lucio@knuth:~$ sudo nmap -sS -sV 192.168.1.100

Starting Nmap 5.21 ( http://nmap.org ) at 2012-10-30 01:56 ART
Nmap scan report for 192.168.1.100
Host is up (0.013s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 5.3p1 Debian 3ubuntu6 (protocol 2.0)
80/tcp open  http?
1 service unrecognized despite returning data.
MAC Address: 00:24:8C:96:57:8B (Asustek Computer)
Service Info: OS: Linux

Service detection performed. Please report any incorrect results
at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 134.77 seconds
```

Ejercicio: conexiones TCP

- En una consola de Linux, ejecutar `watch netstat -an`.
- Abrir otra consola y utilizar `netcat` para escuchar en el puerto 5555 de localhost.
- ¿Qué cambios se observan en la primera consola?
- En otra consola, utilizar nuevamente `netcat` para conectarse a dicho puerto en localhost.
- ¿Qué cambios se observan ahora en la primera consola?
- Cerrar la conexión (presionando Control+D) y analizar en qué estado se la marca en la primera consola.
- Repetir todo lo anterior usando sockets de Python para ambos procesos.

Ejercicio: port scanning

- Utilizar nmap para realizar un SYN scan al gateway de la red local.
- Mientras tanto, abrir Wireshark y observar cómo son los paquetes enviados. ¿Usan el mismo puerto origen? ¿Qué ocurre con el tamaño de la ventana? ¿Y con el número de secuencia inicial?
- Implementar el algoritmo de SYN scan en Scapy y volver a repetir el experimento anterior. Comparar ambas implementaciones a través de los resultados encontrados y sacar conclusiones.

Referencias

- RFCs 768, 1631 y 2663.
- *Nmap Port Scanning Techniques*
  `http://nmap.org/book/man-port-scanning-techniques.html`
- *TCP/IP Fingerprinting Methods Supported by Nmap*
  `http://nmap.org/book/osdetect-methods.html`
- S. McClure, J. Scambray, G. Kurtz (2012)
  Hacking Exposed 7: Network Security Secrets & Solutions - Cap. 2: Scanning
  New York: McGraw-Hill.
- man pages de `telnet` , `netstat` , `netcat` y `nmap` .